2. Implement a client and a server on different computers using python. Perform the authentication

ofsender between these two entities by using RSA digital signature cryptosystem.

Source code :

import socket

from cryptography.hazmat.primitives.asymmetric import rsa, padding

from cryptography.hazmat.primitives import serialization, hashes

# Generate RSA key pair

private\_key = rsa.generate\_private\_key(

public\_exponent=65537,

key\_size=2048,

)

# Get public key in PEM format

public\_key = private\_key.public\_key()

public\_key\_pem = public\_key.public\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo

)

# Create socket and bind to a specific port

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_address = ('localhost', 12345)

sock.bind(server\_address)

# Listen for incoming connections

sock.listen(1)

while True:

print('Waiting for a connection...')

connection, client\_address = sock.accept()

try:

print('Connection from', client\_address)

# Send public key to client

connection.sendall(public\_key\_pem)

# Receive message and signature from client

message = connection.recv(1024)

signature = connection.recv(1024)

# Verify signature using client's public key

client\_public\_key\_pem = connection.recv(1024)

client\_public\_key = serialization.load\_pem\_public\_key(client\_public\_key\_pem)

client\_public\_key.verify(

signature,

message,

padding.PKCS1v15(),

hashes.SHA256()

)

print(f"Received message from client: {message.decode('utf-8')}")

finally:

# Close the connection

connection.close()

Client Code :

import socket

from cryptography.hazmat.primitives.asymmetric import rsa, padding

from cryptography.hazmat.primitives import serialization, hashes

# Create socket and connect to server

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_address = ('localhost', 12345)

sock.connect(server\_address)

try:

# Receive public key from server

public\_key\_pem = sock.recv(1024)

# Load public key from PEM format

public\_key = serialization.load\_pem\_public\_key(public\_key\_pem)

# Create RSA key pair for client

private\_key = rsa.generate\_private\_key(

public\_exponent=65537,

key\_size=2048,

)

# Get public key in PEM format

public\_key\_pem = private\_key.public\_key().public\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo

)

# Send message and signature to server

message = b'Hello, World!'

signature = private\_key.sign(

message,

padding.PKCS1v15(),

hashes.SHA256()

)

sock.sendall(message)

sock.sendall(signature)

sock.sendall(public\_key\_pem)

finally:

# Close the connection

sock.close()

In this implementation, the server generates an RSA key pair and sends the public key to the client. The client generates its own RSA key pair and sends the message and its digital signature to the server. The server verifies the digital signature using the client's public key.

To run this code, you can save the server and client code in two separate Python files, then run them on two different computers. You should make sure that the server is running before running the client code.